WRITE BACK POLICY FOR MEMORY

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/400,391 titled "JSM Protection," filed July 31, 2002, incorporated herein by reference. This application also claims priority to EPO Application No. 03291914.4, filed July 30, 2003 and entitled "Write Back Policy For Memory," incorporated herein by reference. This application also may contain subject matter that may relate to the following commonly assigned co-pending applications incorporated herein by reference: "System And Method To Automatically Stack And Unstack Java Local Variables," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35422 (1962-05401); "Memory Management Of Local Variables," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35423 (1962-05402); "Memory Management Of Local Variables Upon A Change Of Context," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35424 (1962-05403); "A Processor With A Split Stack," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35425(1962-05404); "Using IMPDEP2 For System Commands Related To Java Accelerator Hardware," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35426 (1962-05405); "Test With Immediate And Skip Processor Instruction," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35427 (1962-05406); "Test And Skip Processor Instruction Having At Least One Register Operand," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35248 (1962-05407); "Synchronizing Stack Storage," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35429 (1962-05408); "Methods And

Apparatuses For Managing Memory," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35430 (1962-05409); "Methods And Apparatuses For Managing Memory," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35432 (1962-05411); "Mixed Stack-Based RISC Processor," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35433 (1962-05412); "Processor That Accommodates Multiple Instruction Sets And Multiple Decode Modes," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35434 (1962-05413); "System To Dispatch Several Instructions On Available Hardware Resources," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35444 (1962-05414); "Micro-Sequence Execution In A Processor," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35445 (1962-05415); "Program Counter Adjustment Based On The Detection Of An Instruction Prefix," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35452 (1962-05416); "Reformat Logic To Translate Between A Virtual Address And A Compressed Physical Address," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35460 (1962-05417); "Synchronization Of Processor States," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35461 (1962-05418); "Conditional Garbage Based On Monitoring To Improve Real Time Performance," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35485 (1962-05419); "Inter-Processor Control," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35486 (1962-05420); "Cache Coherency In A Multi-Processor System," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35637 (1962-05421); "Concurrent Task Execution In A Multi-Processor, Single Operating System Environment," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35638 (1962-05422); and "A Multi-Processor Computing System Having A Java Stack Machine And A RISC-Based Processor," Serial No. _____, filed July 31, 2003, Attorney Docket No. TI-35710 (1962-05423).

BACKGROUND OF THE INVENTION

Technical Field of the Invention

[0002] The present invention relates generally to processor based systems and more particularly to memory management techniques for the processor based system.

Background Information

[0003] Many types of electronic devices are battery operated and thus preferably consume as little power as possible. An example is a cellular telephone. Further, it may be desirable to implement various types of multimedia functionality in an electronic device such as a cell phone. Examples of multimedia functionality may include, without limitation, games, audio decoders, digital cameras, etc. It is thus desirable to implement such functionality in an electronic device in a way that, all else being equal, is fast, consumes as little power as possible and requires as little memory as possible. Improvements in this area are desirable.

BRIEF SUMMARY

[0004] Methods and apparatuses are disclosed for managing memory write back. In some embodiments, the method may include examining current and future instructions operating on a stack that exists in memory, determining stack trend information from the instructions, and utilizing the trend information to reduce data traffic between various levels of the memory. As stacked data are written to a cache line in a first level of memory, if future instructions indicate that additional cache lines are required for subsequent write operations within the stack, then the cache line may be written back to a second level of memory. If however, the future instructions manipulate the stack in such a way that no additional cache lines are required for subsequent write operations within the stack, then the first level of memory may avoid writing back the cache line and also may keep it marked as dirty. In this manner, write back from the first level of memory to

the second level of memory may be minimized and overall power consumption may be reduced. Consequently, cache lines containing stack data are preferably written back from first level of memory to a second level of memory once all the words in a cache line have been written to, unless specified otherwise by a stack trend information.

NOTATION AND NOMENCLATURE

[0005] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, semiconductor companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to...." Also, the term "couple" or "couples" is intended to mean either an indirect or direct connection. Thus, if a first device couples to a second device, that connection may be through a direct connection, or through an indirect connection via other devices and connections. The term "allocate" is intended to mean loading data, such that memories may allocate data from other sources such as other memories or storage media.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] For a more detailed description of the preferred embodiments of the present invention, reference will now be made to the accompanying drawings, wherein:

[0007] Figure 1 illustrates a processor based system according to the preferred embodiments;

[0008] Figure 2 illustrates an exemplary controller;

[0009] Figure 3 illustrates an exemplary memory management policy;

[0010] Figure 4 illustrates exemplary decode logic;

[0011] Figure 5 illustrates an exemplary write back policy; and

[0012] Figure 6 illustrates an exemplary embodiment of the system described herein.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0013] The following discussion is directed to various embodiments of the invention. Although one or more of these embodiments may be preferred, the embodiments disclosed should not be interpreted, or otherwise used, as limiting the scope of the disclosure, including the claims, unless otherwise specified. In addition, one skilled in the art will understand that the following description has broad application, and the discussion of any embodiment is meant only to be exemplary of that embodiment, and not intended to intimate that the scope of the disclosure, including the claims, is limited to that embodiment.

[0014] The subject matter disclosed herein is directed to a processor based system comprising multiple levels of memory. The processor based system described herein may be used in a wide variety of electronic systems. One example comprises using the processor based system in a portable, battery-operated cell phone. As the processor executes various system operations, data may be transferred between the processor and the multiple levels of memory, and consequently, memory space within a given level of memory may fill up. In order to provide adequate system operation, data entries in one memory level may need to be written to another level of memory. Furthermore, multiple processor systems also may necessitate transferring data between levels of memory to maintain a coherent data for all processors. In the following discussion, a first level of memory and a second level of memory shared between several processors will be taken as an example. The processor based system may implement a different cache policy based on stack trends and stack data information in order to reduce the number of transfers among the multiple levels of memory. Consequently, the amount of time taken to transfer data between the multiple levels of memory may be reduced and the overall power consumed by the processor based system

may be reduced. In particular, in the embodiment illustrated in Figure 1, a write back policy is implemented in cache controller 26 of system 10, where traditional write back policies are used for most data. In the event that data belongs to specific data structures (i.e., stack data) the cache write back policy is modified in order to maintain within the cache only a single dirty line at a given time as described below.

[0015] Figure 1 illustrates a system 10 comprising a processor 12 coupled to a first level or cache memory 14, a second level or main memory 16, and a disk array 17. The processor 12 comprises a register set 18, decode logic 20, trend logic 21, an address generation unit (AGU) 22, an arithmetic logic unit (ALU) 24, and an optional micro-stack 25. Processor 12 may include a stack based processor, whose instructions operate on a stack. In this manner, Processor 12 may include a "Java Stack Machine" (JSM) as described in the co-pending applications referenced herein. Cache memory 14 comprises a cache controller 26 and a storage space 28.

[0016] Main memory 16 comprises a storage space 30, which may contain contiguous amounts of stored data. For example, if the processor 12 is a stack-based processor, (e.g., JSM) main memory 16 may include a stack 32. Additionally, the stack 32 may also reside in cache memory 14, disk array 17, and/or micro-stack 25. Stack 32 preferably contains data from the processor 12 in a last-in-first-out manner (LIFO). If system 10 comprises a multiple processor system, the stack 32 may need to maintain coherency in a level of memory that is shared between the processors, such as in main memory 16, as described below. Register set 18 may include multiple registers such as general purpose registers, a program counter, and a stack pointer. The stack pointer preferably indicates the top of the stack 32. Data may be added to the stack 32 by "pushing" data at the address indicated by the stack pointer. Likewise, data may be retrieved from the stack 32 by "popping" data from the address indicated by the stack pointer. Also, as will be described below,

selected data from cache memory 14 and main memory 16 may exist in the micro-stack 25. The number of accesses between memory levels and the cost associated with each memory level illustrated in Figure 1 may be adapted to achieve optimal system performance. For example, the cache memory 14 may be part of the same integrated circuit as the processor 12 and main memory 16 may be external to the processor 12. In this manner, the cache memory 14 may have relatively quick access time compared to main memory 16, however, the cost (on a per-bit basis) of cache memory 14 may be greater than the cost of main memory 16.

[0017] As the software executes on system 10, processor 12 may issue effective addresses along with read or write data requests, and these requests may be satisfied by various system components (e.g., cache memory 14, main memory 16, micro-stack 25, or disk array 17) according to a memory mapping function. Although various system components may satisfy read/write requests, the software may be unaware whether the request is satisfied via cache memory 14, main memory 16, micro-stack 25, or disk array 17. Preferably, traffic to and from the processor 12 is in the form of words, where the size of the word may vary depending on the architecture of the system 10. Rather than containing a single word from main memory 16, each entry in cache memory 14 preferably contains multiple words referred to as a "cache line". In this manner, the principle of locality—i.e., within a given period of time, programs tend to reference a relatively confined area of memory repeatedly—may be utilized. Therefore, the efficiency of the multi-level memory may be improved by infrequently writing cache lines from the slower memory (main memory 16) to the quicker memory (cache memory 14), and accessing the cache lines in cache memory 14 as much as possible before replacing a cache line.

[0018] Controller 26 may implement various memory management policies. Figure 2 illustrates an exemplary implementation of cache memory 14 including the controller 26 and the storage space

28. Although some of the Figures may illustrate controller 26 as part of cache memory 14, the location of controller 26, as well as its functional blocks, may be located anywhere within the system 10. Storage space 28 includes a tag memory 36, valid bits 38, dirty bits 39, and multiple data arrays 40. Data arrays 40 contain cache lines, such as CL_0 and CL_1, where each cache line includes multiple data words as shown. Tag memory 36 preferably contains the addresses (or a most significant part of the addresses, depending on the cache associativity) of data stored in the data arrays 40, e.g., $ADDR_0$ and $ADDR_1$ correspond to cache lines CL_0 and CL_1 respectively. Valid bits 38 indicate whether the data stored in the data arrays 40 are valid. For example, cache line CL_0 may be enabled and valid, whereas cache line CL_1 may be disabled and invalid. Dirty bits 39 indicate whether a cache line has been modified. The operation of the dirty bits 39 will be described below with respect to "write back" policies.

[0019] Controller 26 includes compare logic 42 and word select logic 44. The controller 26 may receive an address request 45 from the AGU 22 via an address bus, and data may be transferred between the controller 26 and the ALU 24 via a data bus. The size of address request 45 may vary depending on the architecture of the system 10. Address request 45 may include a upper portion ADDR[H] that indicates which cache line the desired data is located in, and a lower portion ADDR[L] that indicates the desired word within the cache line. Although Figure 2 depicts a fully associative cache 14, this configuration is portrayed solely for the sake of clarity. It should be noted that the subject matter disclosed herein equally applies to any order of multi-way set associative cache structures where the address 45 is split into three parts—an upper portion ADDR[H], a middle portion ADDR[M], and a lower portion ADDR[L]. Compare logic 42 may compare the requested data address to the contents of the tag memory 36. If the requested data address is located in the tag memory 36 and the valid bit 38 associated with the requested data

address is enabled, then the cache line may be provided to the word select logic 44. Word select logic 44 may determine the desired word from within the cache line based on the lower portion of the data address ADDR[L], and the requested data word may be provided to the processor 12 via the data bus. Dirty bits 39 may be used in conjunction with the write back policy. On a write request, if the data is modified within the cache without being modified in main memory, the corresponding dirty bit associated with the cache line where the data resides is set to indicate that the data is not coherent with its value in main memory. Subsequently, dirty cache lines may be evicted from cache space as space is needed. Dirty lines that are evicted during a line replacement caused by a cache miss need to be written back to the main memory, whereas non-dirty lines may be simply overwritten. Consequently, in cache memory structures having write back policies, data are made coherent within the main memory when they are evicted from the cache either when selected as victim line by the replacement policy (e.g. LRU, random), or when an explicit request like a "clean cache" command occurs. The write back policy may be adapted for certain data types in order to improve overall system performance. Decode logic 20 processes the address of the data request and may provide the controller 26 with additional information about the address request. For example, the decode logic 20 may indicate that the requested data address belongs to the stack 32 (illustrated in Figure 1). Using this information, the controller 26 may implement cache management policies that are optimized for stack based operations as described below.

[0020] As a result of the processor 12 pushing and popping data to and from the top of the stack 32, the stack 32 expands and contracts. Data are pushed on the stack 32 and popped off of the top of the stack 32 in a sequential manner—i.e., stack data is preferably accessed using sequential addressing as opposed to random addressing. Also, for the sake of the following discussion, it will be assumed that when the system 10 is addressing stack data, the corresponding address in memory

increases as the stack 32 is growing (e.g. system 10 is pushing a value on to the stack 32). Thus, as stack data is written to new cache lines in cache memory 14, the stack data is written to the first word of this cache line and subsequent stack data are written to the subsequent words of the cache line. For example, in pushing stack data to cache line CL_0 (illustrated in Figure 2), word W_0 would be written to before word W_1. Since data pushed from the processor 12 represents the most recent version of the data in the system 10, consulting main memory 16 on a cache miss is unnecessary.

[0021] In accordance with some embodiments, data may be written to cache memory 14 on a cache miss without allocating or fetching cache lines from main memory 16, as indicated in co-pending application entitled "Methods And Apparatuses For Managing Memory," filed July 31, 2003, serial no. July 31, 2003 (Atty. Docket No.: TI-35430). Figure 3 illustrates a system optimization in the cache management 48 related to a write back policy for stack data that may be implemented by the controller 26. Block 50 illustrates a write request for stack data. As a result of the stack data write request, the AGU 22 may provide the address request 45 to the controller 26. Controller 26 then may determine whether the data is present in cache memory 14, as indicated by block 52. If the data is not present within cache memory 14, a "cache miss" may be generated, and cache memory 14 may allocate a new line per block 54. If the data is present within the cache, a "cache hit" may be generated, the data may be updated, and the dirty bit of the corresponding line may be set as indicated in block 56. During the cache update, the cache controller determines if the address corresponds to the last word of a cache line per block 58. If the address does not correspond to the last word in the cache line, the cache is updated per block 56. Otherwise, if the address corresponds to the last word, the cache controller 26 writes back the line to the main memory as indicated in block 60. When the line is written back to main memory, the dirty bit is cleared indicating that cache and main memory hold coherent data. This creates potentially

additional data transfer compared to a standard write back policy but provides the advantage of reducing the latency when a coherent view of the stack at a lower level of memory is required

[0022] Stack data within the system 10 may need to be written between the multiple levels of memory for several reasons. For example, data arrays 40 (illustrated in Figure 2) may fill up such that cache lines may need to be written back to main memory 16 in order to free up space for new cache lines. Additionally, multi-processor systems may require a coherent view of the stack 32 in main memory 16 since the stack data might need to be read by a second processor (not shown in the figures). Since stack data may exist within the micro-stack 25 and the cache memory 14, maintaining a coherent stack 32 in main memory 16 may involve writing the contents of the micro-stack 25 and the modified cache lines of the cache memory 14 to main memory 16 using write back techniques such as specific cache commands to explicitly write back data from cache memory 14 to main memory 16. Write back techniques may involve dirty bits 39 that indicate the cache lines in cache memory 14 that have been modified but still need to be updated in main memory 16. For example, if word W_0 of cache line CL_0 is modified, then the dirty bit associated with cache line CL_0 may be enabled. In accordance with some embodiments, the number of dirty cache lines in cache memory 14 may be restricted for certain data types (e.g. stack data). For example, cache memory 14 preferably contains a single dirty cache line with stack data. Consequently, the number of cache lines that may need to be written back to main memory 16 from cache memory 14 in order to maintain a coherent stack 32 may be reduced and the latency associated with making the stack 32 coherent in the main memory 16 also may be reduced. With a reduced number of dirty cache lines, system 10 may utilize trend information to avoid unnecessarily writing dirty cache lines back to main memory 16.

[0023] Referring to Figure 4, an instruction pipe 66 of processor 12 is illustrated including instructions $INST_0$ through $INST_N$. In a processor there may be several stages between the execution of a data memory access and the decode of an instruction. In addition, these stages might include some predecoding stages, such that the information supplied to the trend logic 21 might come from some decode and predecode logic. In the preferred embodiment, the memory access stage and the decode stage are separated by several pipe stages. Accordingly, the trend logic 21 may use one or more signals supplied by the decoder logic 20. As instructions progress within the pipe their effect on the stack 32 is taken into account by the trend logic to generate the appropriate trend information to the cache controller. For example, trend logic 21 may receive CURRENT 68 and FUTURE 70 to generate the trend information about the stack 32. When the instruction preceding instruction $INST_0$ is accessing memory, the trend logic 21 receives information from decode logic 20 and sends to the cache controller CURRENT 68 stack trend information relating to what effect instruction $INST_0$ will have on the stack 32. In addition, the trend logic may send information indicating FUTURE 70 stack trend information corresponding to the following instruction $INST_1$ through $INST_N$. For example, if $INST_0$ is an "iload" instruction (which pushes an operand on to the stack 32), then this may cause the stack 32 to increase by 1, whereas if $INST_0$ is an "iadd" instruction (which pops two operands from the stack adds them together and pushes the result back on the stack), then this may cause the stack 32 to decrease by 1. In a similar fashion, FUTURE 70 represents signal coming from the decoder 20 and potentially a predecode logic corresponding to a predetermined number of instructions following $INST_0$ within the instruction pipe 66 (e.g., $INST_1$ through $INST_N$), and providing information on the evolution of the stack for the subsequent instructions to trend logic 21. Trend logic 21 may then utilize the CURRENT 68 and FUTURE 70 stack trend information to determine a net stack trend. For example, if $INST_1$

increases the stack by 1, $INST_2$ decreases the stack by 1, and $INST_3$ decreases the stack by 2, then the net stack trend is to be decreased by 2. Using the net stack trend, the trend logic 21 may maintain more than one dirty cache line in cache memory 14 where the trend information may indicate that some of the data in the dirty cache lines are going to be consumed by the subsequent instructions.

[0024] Referring to Figure 5, a write back policy 72, which takes into account trend information, is illustrated that may be implemented by controller 26. Block 74 illustrates a write request coming from system 10 as a result of a micro-stack overflow. The micro-stack overflow itself results from the instruction currently executing a data write access within the stack. As a result of the write request, the AGU 22 may provide the write request 45 to the controller 26. Controller 26 then may determine whether the write request is going to write to the end of the single dirty cache line in cache memory 14, as indicated by block 76. If the write request is not directed to the end of the dirty cache line, then the write request will be performed. For example, if cache line CL_0 (illustrated in Figure 2) represents the cache line in which stack data are currently written in the cache memory 14, and the stack pointer indicates that word W_1 is going to be written to, then the write request will be performed to cache line CL_0. Subsequent write requests also may write to cache line CL_0 until the end of the cache line is written to, i.e., stack pointer indicates that word W_N is going to be written to. If the last word W_N of a cache line holding stack data is going to be written to, then the overall stack trend may be evaluated as illustrated by block 80. If the overall stack trend is increasing, then subsequent write operations may require another cache line because all of the words in the current dirty cache line have been written to. Accordingly, to maintain a single dirty cache line within the cache memory 14, the dirty cache line may be written to main memory 16 if the stack trend is increasing as indicated by block 82. If the overall stack trend is

decreasing, data within this line are going to be consumed next. Consequently, writing data from the dirty cache line to main memory is unnecessary. As illustrated in block 84, the cache line is kept in the cache memory 14 until the stack trend information indicates that the future instructions will cause the stack to increase beyond the last word in the dirty cache line.

[0025] Although the embodiments refer to situations where the stack 32 is increasing, i.e., the stack pointer incrementing as data are pushed onto the stack 32, the above discussion equally applies to situations where the stack 32 is decreasing, i.e., stack pointer decrementing as data are pushed onto the stack 32. Also, instead of checking of the last word of the cache line during the cache to adapt the cache policy, checking of the first word of the cache line may be performed. For example, if the stack pointer on a write access is referring to word W_0 of a cache line CL_0, and the trend information indicates that the stack is going to decrease on the following instructions, then the currently written line is kept dirty within the cache and write back to the main memory is avoided.

[0026] As was described above, stack based operations, such as pushing and popping data, may result in cache misses. The micro-stack 25 may initiate the data stack transfer between system 10 and the cache memory 14 that have been described above as write access on stack data. For example, in the event of an overflow or underflow operation, as is described in copending application entitled "A Processor with a Split Stack," filed July 31, 2003, serial no. ___ (Atty. Docket No.: TI-35425) and incorporated herein by reference, the micro-stack 25 may push and pop data from the stack 32.

[0027] As noted previously, system 10 may be implemented as a mobile cell phone such as that illustrated in Figure 4. As shown, a mobile communication device includes an integrated keypad 412 and display 414. The processor 12 and other components may be included in electronics

package 410 connected to the keypad 412, display 414, and radio frequency ("RF") circuitry 416. The RF circuitry 416 may be connected to an antenna 418.

[0028] While the preferred embodiments of the present invention have been shown and described, modifications thereof can be made by one skilled in the art without departing from the spirit and teachings of the invention. The embodiments described herein are exemplary only, and are not intended to be limiting. Many variations and modifications of the invention disclosed herein are possible and are within the scope of the invention. For example, the various portions of the processor based system may exist on a single integrated circuit or as multiple integrated circuits. Also, the various memories disclosed may include other types of storage media such as disk array 17, which may comprise multiple hard drives. Accordingly, the scope of protection is not limited by the description set out above. Each and every claim is incorporated into the specification as an embodiment of the present invention.